**AI Project**

**Multiple object recognition in an image**

Object detection is an important task in today’s world where everyone of us require not only what all objects are present in an image( which is object recognition) but we also require where it is present in that image. That too, if there are multiple objects present in an image that is to be detected, then the task becomes much more complicated because of several factors like occlusion, intensity variations etc;

So, our project is to implement a model that can detect multiple objects in an image.

**Implementation steps:**

1. Collection of data and processing it to have it in required format
2. Creating a model in keras, tensorflow and training that with this data
3. Give test data to check and according to the accuracy values improve the training/ fine tune parameters

**Data flow:**

Input image → Model → Output image with bounding boxes of objects and probabilities

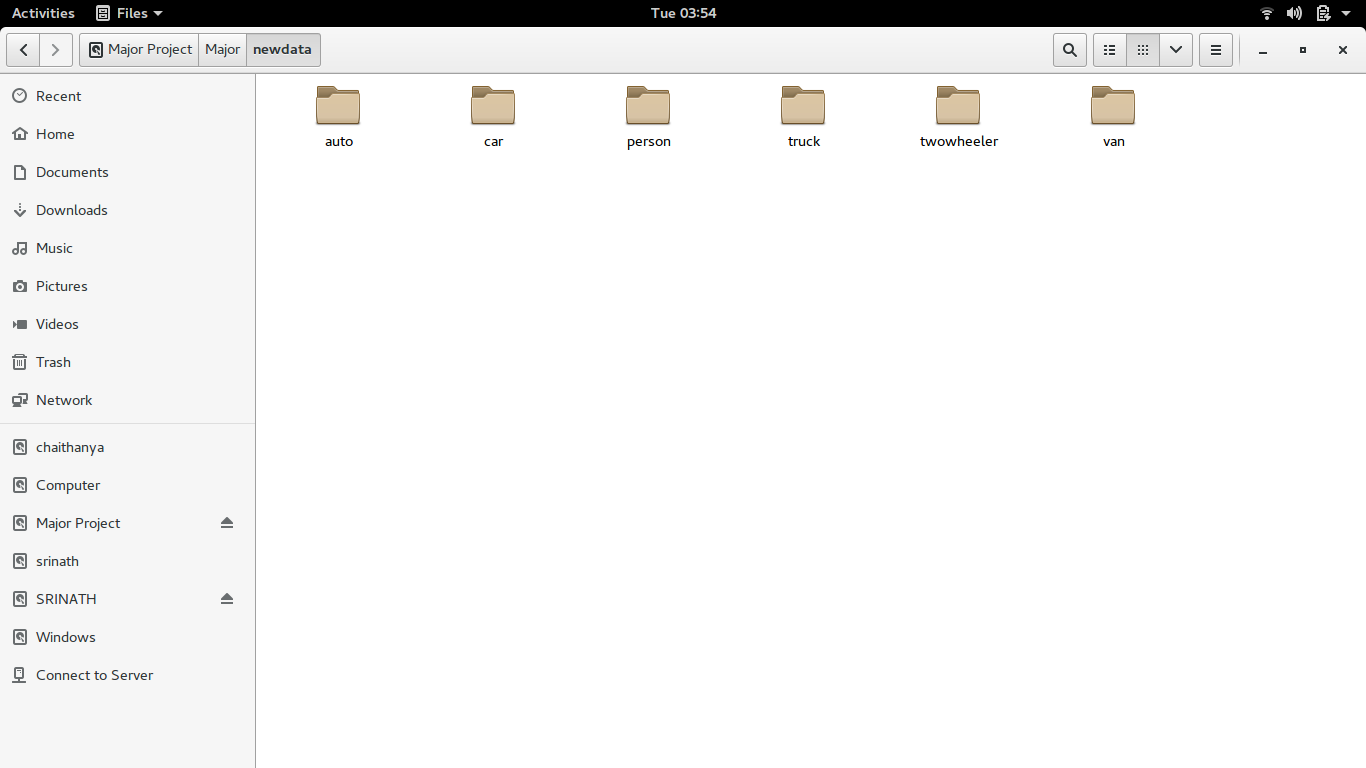
**Neural networks to be used:**

As we will be working on images, CNN will be used. In particular we are planning to use MobileNet because of its speed in detection and reduction of space consumption[1].

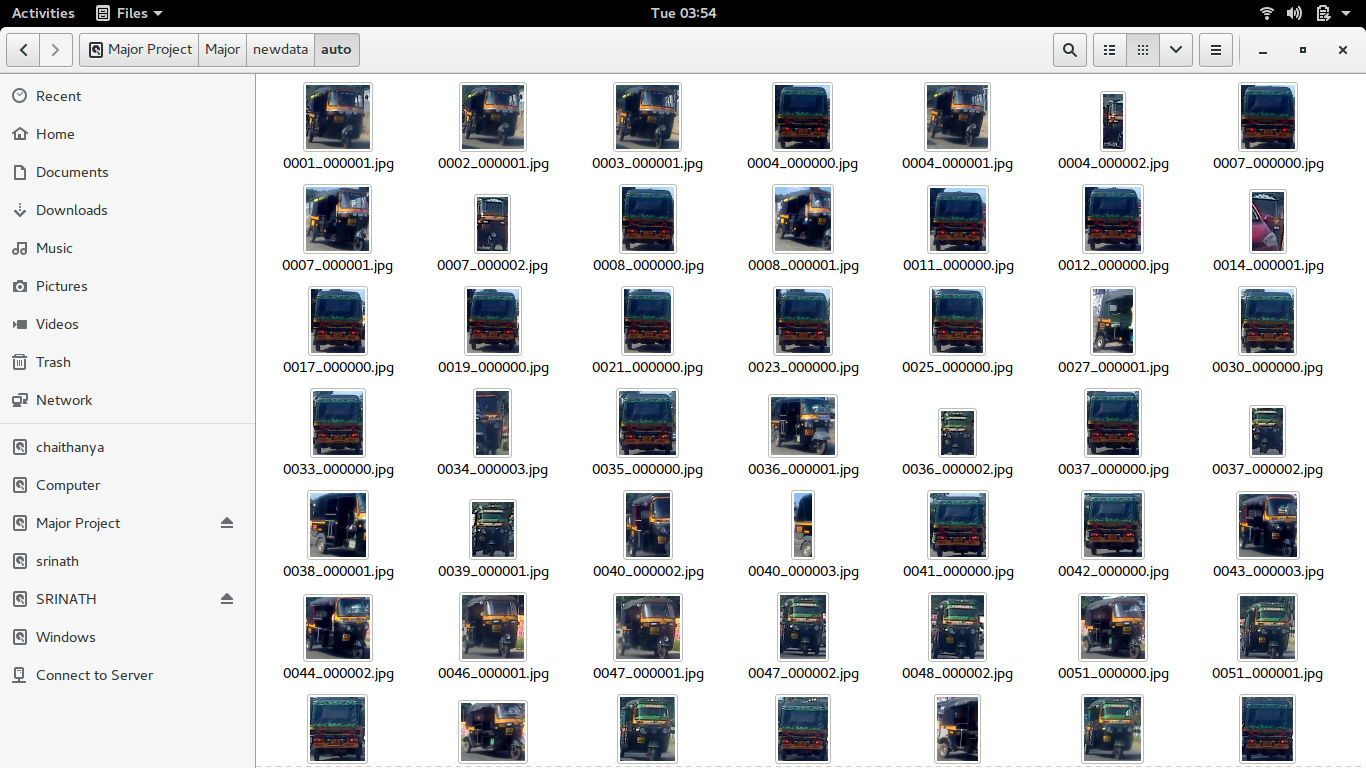
**Proposed Step 1:**

**Collection of data and preprocessing:**

1. Instead of using any standard dataset, we created our own dataset for object detection on Indian roads. So, there are 6 classes namely Auto, Car, Two Wheeler, Truck, Van and Pedestrian. A camera is mounted infront of car and the video of road is recorded. Now, this video is divided into images by fps(Frame per second) and are saved. Objects present in each image are marked by using “Labelbox” and online application which outputs the coordinates after manual labelling in json, csv formats. 1200 images are labelled.
2. Data to be given to mobilenet should be in particular format i.e images corresponding to particular class should be in a folder with the name of the folder same as the class name. For this, the json file is accessed and a python script**(crop.py)** is written which calls each and every image present in a folder, checks what is the object(based on json data) and crops that section of image and saves into another folder of our choice.



**How data is to be prepared for this network**



**Inside each folder, their corresponding cropped images are to be placed**

**Proposed Step 2:**

**Creating a model in keras, tensorflow and training:**

1. Mobilenet architecture pretrained with Imagenet data is available in keras.
2. When the basic architecture is loaded and trained, it took nearly 3hrs to complete 5epochs. The reason for such a long time is the inefficiency of tensorflow, keras to use GPU(Graphical Processing Unit). As the system has NVIDIA drivers, the tensorflow, CUDA and CuDNN have been installed properly with compatible versions such that tensorflow uses GPU. After this step, the training took about 5min.
3. Now the model is trained and an accuracy of 98% is attained within 5 epochs. But when the test images are given, its prediction rate is very bad.
4. Looking into the reasons, it was found that the loading of images to the model architecture are to be done in a particular format i.e the images before loading to the first layer of CNN should be preprocessed, reshaped according to the specifications of architecture by using functions of keras. But till then it is loaded by using Opencv. So, this step has been rectified and the prediction accuracy is high.

**Proposed Step 3:**

**Give test data to check and according to the accuracy values improve the training/ fine tune parameters**

1. The issue now persisting is the model can detect whether the given image is “Auto” or “Pedestrian” but it cannot localize and detect i.e cant predict the bounding box coordinates.
2. To achieve this, the last few layers has to be changed such that instead of just giving the probabilities of the classes, it also gives the coordinates, confidence scores etc;
3. Also, the model may predict many bounding boxes out of which the best one is to be choosen. For that the concepts of IoU(Intersection over Union) and NMS(Non Max Suppression) has to be coded.
4. It was mentioned in [1] that object detection using mobilenet is achieved by changing the last layers to FR-CNN(Faster R-CNN), SSD(Single Shot Detection) or YOLO(You Only LOOk Once)
5. So, these concepts couldn’t be implemented in code within time.
6. Instead, another architecture named YOLO is directly used which has the final layers adjusted to output the coordinates.
7. YOLO[2] was implemented using “darknet” a GitHub repository.

**Code explanation:**

1. As the crop.py is too large, it will be included in GitHub repository and the link will be provided.

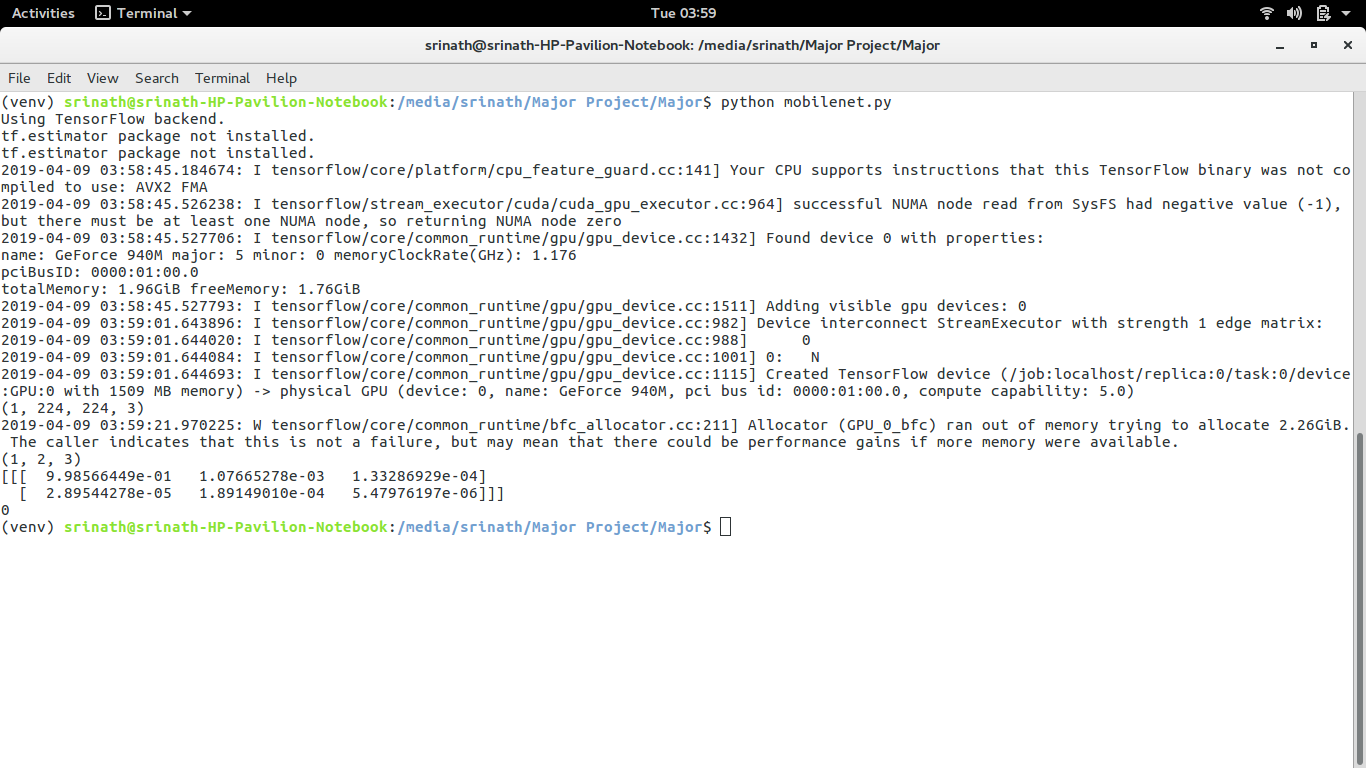
**mobilenet.py**

1. import pandas as pd
2. import numpy as np
3. import os
4. import keras
5. import matplotlib.pyplot as plt
6. from keras.layers import Dense,GlobalAveragePooling2D
7. from keras.applications import MobileNet
8. from keras.preprocessing import image
9. from keras.applications.mobilenet import preprocess\_input
10. from keras.preprocessing.image import ImageDataGenerator
11. from keras.models import Model, load\_model
12. from keras.optimizers import Adam
13. import cv2
14. import tensorflow as tf
15. #os.environ["CUDA\_DEVICE\_ORDER"] = "PCI\_BUS\_ID"
16. #os.environ["CUDA\_VISIBLE\_DEVICES"] = "0,1,2,3"
17. # from keras import backend as K
18. # print(K.tensorflow\_backend.\_get\_available\_gpus())
19. #sess = tf.Session(config=tf.ConfigProto(log\_device\_placement=True))
20. # In[2]:
21. # import tensorflow as tf
22. # with tf.device('/gpu:0'):
23. # a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
24. # b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
25. # c = tf.matmul(a, b)
26. # with tf.Session() as sess:
27. # print (sess.run(c))
28. # base\_model=MobileNet(weights='imagenet',include\_top=False) #imports the mobilenet model and discards the last 1000 neuron layer.
29. # x=base\_model.output
30. # x=GlobalAveragePooling2D()(x)
31. # x=Dense(1024,activation='relu')(x) #we add dense layers so that the model can learn more complex functions and classify for better results.
32. # x=Dense(1024,activation='relu')(x) #dense layer 2
33. # x=Dense(512,activation='relu')(x) #dense layer 3
34. # preds=Dense(6,activation='softmax')(x) #final layer with softmax activation
35. # # In[3]:
36. # model=Model(inputs=base\_model.input,outputs=preds)
37. #specify the inputs
38. #specify the outputs
39. #now a model has been created based on our architecture
40. # In[4]:
41. for layer in base\_model.layers[:20]:
42. layer.trainable=False
43. for layer in base\_model.layers[20:]:
44. layer.trainable=True
45. # In[5]:
46. model = load\_model("/media/srinath/Major Project/Major/newmobilenet.h5")
47. # train\_datagen=ImageDataGenerator(preprocessing\_function=preprocess\_input) #included in our dependencies
48. # train\_generator=train\_datagen.flow\_from\_directory('/media/srinath/Major Project/Major/newdata', # this is where you specify the path to the main data folder
49. # target\_size=(224,224),
50. # color\_mode='rgb',
51. # batch\_size=32,
52. # class\_mode='categorical',
53. # shuffle=True)
54. # In[33]:
55. #model.compile(optimizer='Adam',loss='categorical\_crossentropy',metrics=['accuracy'])
56. # Adam optimizer
57. # loss function will be categorical cross entropy
58. # evaluation metric will be accuracy
59. # step\_size\_train=train\_generator.n//train\_generator.batch\_size
60. # model.fit\_generator(generator=train\_generator,
61. # steps\_per\_epoch=step\_size\_train,
62. # epochs=3)
63. #model.summary()
64. #print(model.summary())
65. #model.save('/media/srinath/Major Project/Major/newmobilenet.h5')
66. img = image.load\_img("auto.jpeg", target\_size = (224,224))
67. test\_image = image.img\_to\_array(img)
68. test\_image = np.expand\_dims(test\_image, axis = 0)
69. test\_image =preprocess\_input(test\_image)
70. print(test\_image.shape)
71. output = model.predict(test\_image)
72. output = output.reshape(len(output), 2, -1)
73. print(output.shape)
74. print(output)
75. print(output.argmax())
76. # pred\_bboxes =output[...,4]\*224
77. # pred\_shapes = output[..., 4:5]
78. First all the necessary libraries are imported (from line 1-14)
79. Now, from line 15-27 is used to check whether the tensorflow is installed and working in GPU
80. Line 28-44 is for model architecture and freezing of the first few layers for training. Because all the important features are taken in hidden layers
81. Line 46-53 is for importing the input images. Some hyperparameters are defined here.
82. Line 55-64 describe the compiling, fitting and printing the summary of the model
83. Line 65 is used to save the model so that another training is not required
84. Line 66-75 is for predicting output classes.

**Results**



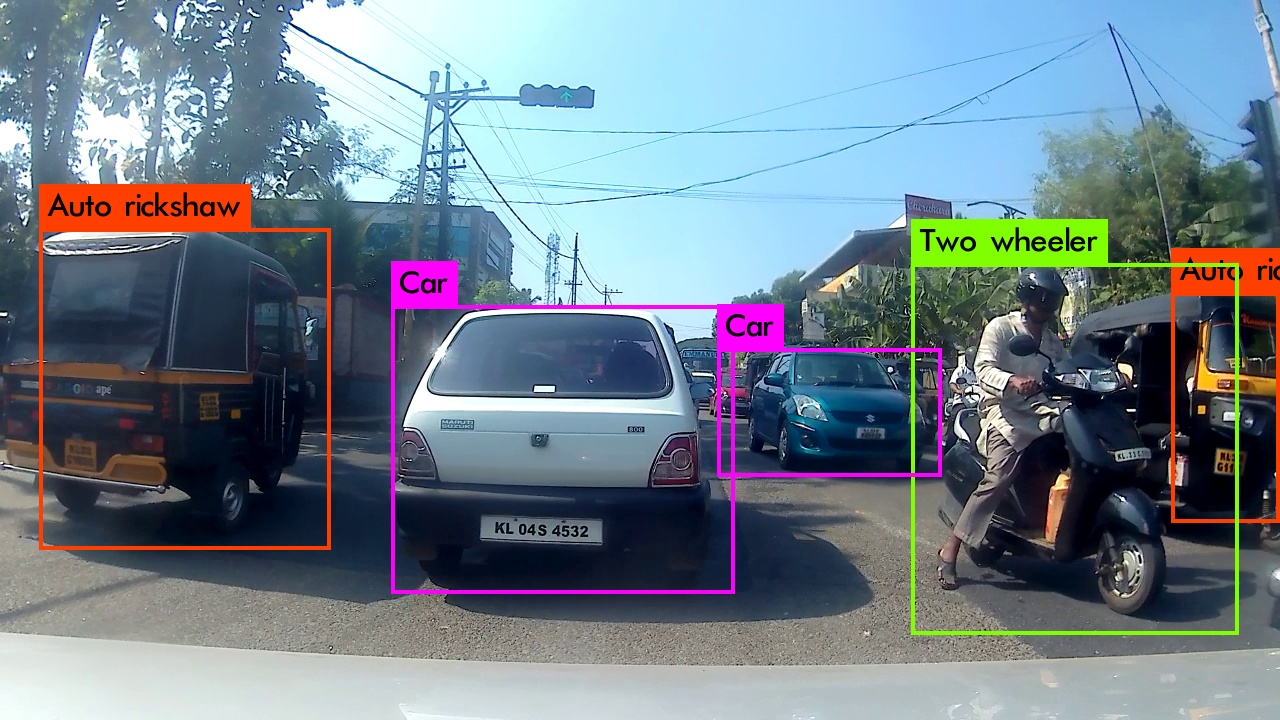
**Input image given to the trained model. All the trained autos are black but the model can detect an yellow one.**



**Output obtained for this image for MOBILENET ARCHITECTURE. Class 0 indicates it is an auto. The vector gives the probabilities of each class**



**Another example. But for this the model predicted as truck because the data has several trucks which are in blue color**



**Output of YOLO model. It can detect multiple objects present in an image with their probabilities and coordinates**

**What has been learned by this project:**

1. Processing of images and moving through the folders to auto save the cropped images using json file
2. Using of GPU to train a model. Installing of CUDA, CuDNN and Tensorflow
3. Building the model and fine tuning
4. Input a test image properly to the model
5. Working with GitHub repository and building YOLO models.
6. Debugging errors related to version compatibility, image formatting etc;

**References:**

**[1]** A.Howard. *Mobilenets: Efficient convolutional neural networks for mobile vision applications.*

**[2]** Redmon, Joseph, et al. *"You only look once: Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.*